# Distributed Intelligent Planning and Scheduling (DIPS)

Subrata Das, Dan Knights, Curt Wu
Charles River Analytics, Inc.
725 Concord Avenue
Cambridge, MA 02138, USA
+1 617 491 3474
{sdas, dknights, cwu}@cra.com

Walt Truszkowski
NASA Goddard Space Flight Center
Code 588
Greenbelt, MD 20771, USA
+1 301 286 8821
Walt.Truszkowski@gsfc.nasa.gov

## ABSTRACT

In this paper, we present a system for Distributed Intelligent Planning and Scheduling (DIPS) that helps a spacecraft function as an autonomous agent. A DIPS-based spacecraft receives only high-level goals from ground station operators, and performs its own planning and scheduling onboard, achieving these goals with onboard subsystems and in cooperation with other spacecraft. The task decentralization in DIPS employs a domain distribution algorithm that typically creates a feasible schedule after the first coordination effort, thereby decreasing inter-agent negotiation during the scheduling process. The reasoning performed by DIPS agents to optimize time and resource usage while maintaining flight rules and constraints is based on a constraint propagation paradigm. Priority-based scheduling is implemented, and a hierarchical inter-agent confirmation/authorization system is used for global goal coordination. An enhanced prototype is developed and demonstrated using space-based scenarios involving onboard instruments and a satellite constellation. The vertically layered architecture of the DIPS prototype integrates: 1) Java-based agent inference engine; 2) Prolog platform SICStus for constraint-based reasoning; and 3) KQML for inter-agent communication. We are specifically targeting our effort to enhance the planning and scheduling capability of NASA's planned nanosatellite constellations.

## Keywords

Multi-agent system, planning, scheduling, communication, coordination, constraint propagation, KQML, and Prolog.

## 1. INTRODUCTION

Spacecraft autonomy has the potential for effecting significant cost savings in mission operations by reducing the need for dedicated ground staff. In an autonomous operating mode, operators will communicate only high-level goals and deadlines directly to the spacecraft. The spacecraft will then perform its own planning and scheduling by decomposing a goal into a set of sub-goals to be achieved in cooperation with other spacecraft in the environment. In this paper, we present the DIPS system that helps a spacecraft function as an autonomous agent by incorporating this distributed approach to onboard planning and scheduling.

The term *planning* refers to the generation of activities that satisfy a current set of goals. For example, a planning process to satisfy the request for an image generates activities such as rolling the camera to the correct position and activating the camera shutter. The term *schedule* describes an association of these specific activities with particular start and end times by satisfying temporal constraints (e.g., rolling should be performed before the shutter action). The onboard spacecraft subsystems must execute these time-sensitive activities autonomously to achieve the goals, and if none of the subsystems of the spacecraft is capable of executing an activity then cooperation from another spacecraft in the environment is required.

Two major trends for task representation in the history of AI planning have been observed (Georgeff 1987): goal achievement (GA) and hierarchical task network (HTN). The origin of GA-based planning is in STRIPS (Fikes 1971). In this model of representation, an initial situation, a set of possible actions, and a goal that is to be achieved are given. Planning consists of finding a sequence of actions that would lead from the initial situation to the final one. Several planners were subsequently built on the GA model including TWEAK (Chapman 1987), and SNLP (McAllester 1994). In a planner based on the HTN representation, which originated with NOAH (Sacerdoti 1974), planning proceeds by selecting a non-primitive task, decomposing it into subtasks using a library of available decomposition methods, and then detecting and resolving conflicts with other tasks. This process is repeated until no non-primitive tasks remain and all the conflicts have been resolved. Typical examples of HTN planners are FORBIN (Dean 1988), and NONLIN (Tate 1977). There are also planners combining features from these two such as O-Plan (Currie 1991) and SIPE (Wilkins 1988).

Given a representation in either GA or HTN, solving a planning problem can be viewed as a straightforward search problem, but in general the HTN paradigm can lead to more efficient planners because it allows the user to limit the search space by guiding the

planner towards acceptable solutions. A typical implementation of the search engine of a planner operates on a temporal database such as the HSTS system (Muscettola 1994) and Time Map Manager (Boddy 1994). The search engine posts constraints to the database. The temporal database then constructs a constraint network and provides a constraint propagation service (LePape 1990) to verify the global consistency of the posted constraints with the goals, rules and constraints of the spacecraft. Both the consistency checking and the search for an optimal solution in cooperation with other agents in the environment are computationally intractable, that is, NP-hard. A distributed approach to planning and scheduling allows cooperation among agents in the environment and increases efficiency in the search for an optimal solution by partitioning the whole search space.

In recent years, there has been a growing interest in agent-oriented problem solving (CACM 1994), which provides the basis of our proposed distributed solution (Chaib-draa 1992) to planning and scheduling. The agent-oriented problem-solving environment increases efficiency and capability (Rosenschein 1982) by employing a set of agents that communicate and cooperate with each other to achieve their goals. We use the term *agent* to refer to an entity that operates autonomously or semi-autonomously while interacting with other agents in the environment by means of communication. Although types of agents range from software agents (Genesereth and Ketchpel 1994) implementing the behavior of humans, machines or hardware, to mechanical or electronic robots (Simmons 1991) with the capability of perceiving or sensing the environment and executing appropriate actions, our assumption is that every agent will have an interface that understands a common communication language.

In our envisioned distributed (or equivalently, *multi-agent*) environment (Conry 1988; Georgeff 1983), a set of problem-solving autonomous agents (spacecraft and onboard subsystems of a spacecraft) based on DIPS communicate, cooperate, and negotiate to achieve high-level goals through planning and scheduling. Distributed planning and scheduling emphasizes a decentralized organization in which schedules are generated and executed cooperatively and concurrently by agents. This can be contrasted with a centralized planning environment in which goals, rules, and constraints from individual agents are accumulated at a central place, and a centralized planner is used to generate a global schedule. The centralized approach is particularly unsuitable when the problem is inherently distributed such as in a spacecraft environment where each subsystem or spacecraft functions autonomously.

The domain knowledge of tasks and their components in DIPS are manifested through a hierarchical language taking into account spacecraft operational aspects and resource constraints. The task decentralization in DIPS is performed by employing a domain distribution algorithm that typically allows a feasible schedule to be found after only the first coordination effort, therefore greatly decreasing the need for inter-agent communication during the scheduling process. The reasoning performed by DIPS agents for scheduling tasks by optimizing time and resources is based on a constraint propagation paradigm. An enhanced prototype has been developed and demonstrated using space-based scenarios involving onboard sensors and a satellite constellation. The vertically layered architecture of the DIPS prototype integrates: 1) Java-based agent inference engine; 2) Prolog platform SICStus for constraint-based reasoning; and 3) KQML for inter-agent communication.

The rest of the paper is organized as follows. First we describe a space-based scenario to illustrate the envisioned operating mode of a spacecraft agent: to achieve high level goals through distributed planning and scheduling. Then we present the DIPS architecture in Section 3 that can be instantiated appropriately to implement an agent in the environment. The hierarchical syntax for modeling an agent's domain knowledge of tasks is presented in Section 4. Section 5 describes the protocol for inter-agent communication. Section 6 contains our approach to decentralization and coordination of tasks among agents. The functionality of the current prototype is described in Section 7. Finally, we summarize our work in Section 8 and lay out our future plan for extending the work.

## 2. SATELLITE CONSTELLATION SCENARIO

We present a scenario that will illustrate our envisioned distributed planning and scheduling by incorporating several key problem areas prevalent in a distributed scheduling environment. The simulation of this environment consists of a constellation of satellites, each with a number of local resources and the knowledge of hierarchical task decompositions.

The primary goal of the envisioned distributed system is to successfully distribute high-level task requests to multiple independent resources and to maintain the consistency of constraints placed on those requests. In trying to fulfill this goal, however, there are several problems that can arise. These include replanning due to over-scheduling of a resource, forced scheduling of high priority requests, increased priority based on a request's Time To Live (TTL), accommodation of lapses in agent communication/availability, negotiation with competitive satellites, resource/property management, and use of idle time for schedule optimization.

### *2.1.1 Scheduling Environment*

The scenario consists of a constellation of low-Earth orbiting spacecraft that have cross-link communication capability, each carrying nearly the same suite of instruments. Controlling the constellation is done via a futuristic network of ground stations, in which each ground station can communicate to only one satellite at a time. Each onboard instrument is locally controlled; requests can be sent to the network from any ground station. A request is defined as a goal that requires use of an instrument and associated data memory storage, containing start/stop times and operational

parameters. Agents can communicate with each other in order to coordinate high-level goals and to maintain global constraints.

### *2.1.2 Scenario Hierarchy Specification*

The Satellite Constellation Scenario has the following hierarchy:

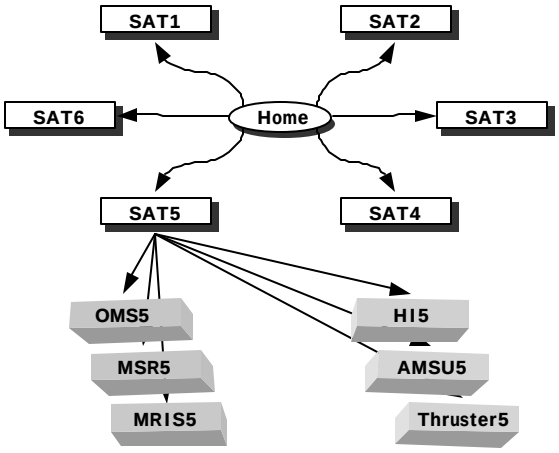



**Figure 1: Scenario Hierarchy**

?? Home (Ground control)—This represents a ground control station on Earth. It has 6 satellite children, "SAT1" through "SAT6".

?? SAT1-SAT6: Each of these DIPS-based agents represents a satellite in the constellation. Every satellite has essentially the same collection of instruments onboard with slight variations in capability/efficiency.

?? Onboard Resources: Ozone Mapping Spectrometer (OMS), Microwave Scanning Radiometer (MRS), Moderate Resolution Imaging Spectroradiometer (MRIS), Hyperspectral Imager (HI), Advanced Microwave Sounding Unit (AMSU), Thrusters Agent.

Each resource onboard a DIPS Satellite Agent has its own local DIPS Resource Agent. There may also be some intermediate DIPS System Agents introduced on each satellite in order to manage groups of related resources.

### *2.1.3 Basic Distributed Scheduling*

In order to demonstrate basic distributed scheduling capabilities, the scenario begins with a number of high-level goal requests introduced to the constellation by the "Home" ground control agent. Every capable DIPS SAT Agent decomposes the task, and corresponding subtask requests are sent to onboard resource and system agents. The scenario includes multiple compound task requests with overlapping time domains. The DIPS-based Agents perform a reasonable effort of coordinating these task distributions so that most are successfully scheduled on the first pass without the need for rescheduling or inter-agent negotiation.

### *2.1.4 Special Situations*

In order to force over-scheduling on some local onboard resources, the scenario includes several conflicting requests. Every DIPS SAT Agent will prefer to schedule its subtasks on the least expensive resources available—sometimes at the expense of over-scheduling a preferred local resource with tentative subtask requests. Top priority requests are introduced to the constellation as forced requests, which will always succeed and be locked into the necessary local schedules, even if other non-forced tasks have been scheduled on those resources at conflicting times.

As a task nears its execution time, its Time To Live (TTL) decreases. A DIPS-based agent will process a task request with a very short TTL before another task request with a long TTL, even if the latter is of higher priority.

The scenario also includes temporary lapses in communication between specific satellites and other DIPS agents in the community. A satellite will move out of the range of a ground station for a specified time period, during which a message relay system using other agents in the community may still allow messages to reach their destination. For example, every DIPS agent that receives a task request addressed to "ALL" immediately forwards the request to all of its siblings in the community to ensure that the requests reach all potential executors.

It is conceivable that a DIPS agent may wish to negotiate with non-friendly satellites in a competitive environment. The scenario includes at least one situation that requires a satellite to use a more expensive competitor satellite in order to complete a task when a request cannot be accommodated locally.

An important real-life issue in a satellite constellation is the management of resources and physical properties. For example, an image of a certain region of Earth can only be taken within a certain time frame while a satellite is over that location. There may also be certain constraints on power consumption that affect resource availability. The scenario includes several task requests that require a DIPS Satellite Agent to reason about its resources and physical properties. Satellites also are required to perform certain mandatory maintenance tasks throughout the scenario.

The scenario allows some idle time for each satellite during which it may explore other feasible solutions to the problem in order to optimize its schedule and accommodate as many task requests as possible. If all requests have been successfully decomposed, a satellite can use its idle time to search for more optimal usage of time and resources.

## 3. DIPS AGENT ARCHITECTURE

The architecture of a DIPS agent is deliberative: there is an explicit symbolic representation of the model of the dynamic environment, and DIPS agents make decisions via logical reasoning based on pattern matching and symbolic manipulation. Several different deliberative agent architectures have been proposed in the literature, and two of them are most prominent: *horizontally layered architecture* (Ferguson 1992) and *vertically layered architecture* (Muller 1994). Either layered approach models an agent as consisting of several hierarchical functional modules representing different requirements of an agent, possibly

incorporating communication, reaction, inference for planning or scheduling, perception, knowledge maintenance, etc. Each layer in a horizontally layered architecture has access to both the perception and the action components whereas in a vertical approach, only one layer has direct interface to the perception and action.

The architecture we have adopted is displayed in Figure 2 and it fits into the vertically layered category. The three layers are the world interface layer, the inference layer, and the network management layer. A DIPS agent's knowledge base is also split into three modules corresponding to the three layers.

## 3.1 World Interface Layer

The *world interface layer* contains a DIPS agent's facilities for *perception, action, and communication*, which all require a detailed knowledge about the environment. A DIPS agent's world model contains information about the environment such as the locations and capabilities of other agents. The world interface layer enables a DIPS agent to communicate with other agents in the environment and perform activities related to planning and scheduling such as sending and receiving requests, responding to a request, etc.

### *3.1.1 Action and Perception*

The *action* and *perception* facilities can be handled through an advanced real-time scripting language. A DIPS agent's actions will be performed via calls to scripts that interact with local hardware. We are exploring integration with the Spacecraft Command Language (SCL) developed by Interface & Control Systems (ICS 1999). SCL, with its innovative Real-Time Engine (RTE), is especially well suited for our real-time scheduling application. The RTE supports both time- and event-based script scheduling as well as real-time resource monitoring and exception handling.

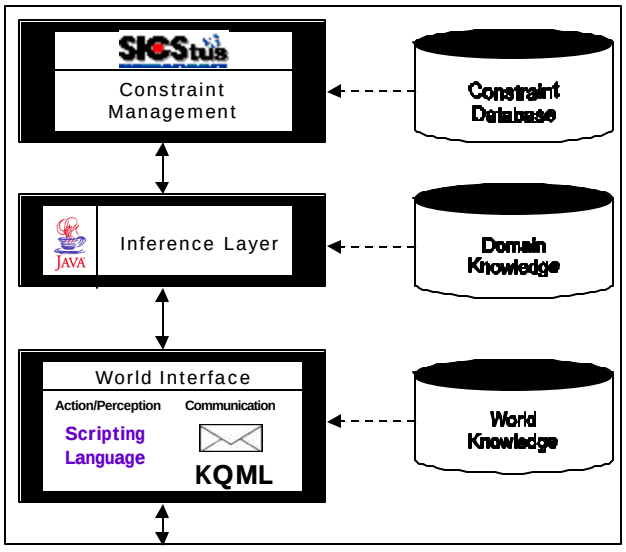



**Figure 2: Vertically Layered DIPS Agent Architecture**

Through such an interaction, schedules generated by the *inference layer* could be actuated on local resources. The *world knowledge*, or information about the states of these resources, may be gathered through sensor data. The *inference layer* can access this through the scripting language, which provides the *perception* component of the *world interface*.

### *3.1.2 Communication*

All inter-agent communication and knowledge manipulation is done via message passing. A DIPS agent uses messages in Knowledge Query and Manipulation Language (KQML) compliant format to communicate goal requests, goal confirmations/denials, capability insertions, and other standard KQML performatives. The *world interface layer* sends and receives messages from other agents in the community and passes them to the *inference layer* to be handled appropriately.

## 3.2 Inference Layer

Written entirely in Java, the *inference layer* of the DIPS prototype agent applies its *domain knowledge* to goal-related messages in order to decompose compound task requests or to schedule primitive tasks on local resources.

The domain knowledge consists of the knowledge of the application such as definitions of different task abstractions and the effects of a task when it is carried out. Although most of the domain knowledge is static for the duration of a particular application, it can still be manipulated at runtime through message passing. When a new resource comes online, for example, any new relevant scripts or task definitions can be added to a DIPS agent's domain knowledge through capability insertions. The content of the domain knowledge and the functionality of a DIPS agent's inference layer depend on the type of agent. The DIPS system currently recognizes two distinct subclasses within the DIPS agent architecture: System Agents and Resource Agents.

### *3.2.1 DIPS System Agent*

A DIPS *System Agent* represents a system of primitive resources, so the capabilities of a DIPS System Agent (e.g., a Satellite) consist only of decomposable compound tasks. A DIPS System Agent does not maintain a local schedule for it has no local resources; rather it coordinates a group of subagents, some of which may also be System Agents. When a DIPS System Agent receives a high-level goal request, it uses a predefined decomposition from its domain knowledge to create a plan for that task, and then uses its *constraint management layer* to create a feasible schedule for that plan. Each subtask in the plan is assigned a certain time interval based on the decomposition constraints, and subtask requests are sent out to all subagents.

### *3.2.2 DIPS Resource Agent*

A DIPS *Resource Agent* represents a physical onboard resource. Its capabilities are all primitive tasks that can be performed via calls to predefined low-level scripts. When a DIPS Resource Agent receives a primitive task request, it posts new temporal constraints describing the task to the *constraint database*. The

inference layer uses the constraint management layer to verify the feasibility of its new augmented local schedule and to generate an instance if feasible.

## 3.3 Constraint Management Layer

The *constraint management layer* of a DIPS agent is based in a version of the Prolog language called SICStus Prolog, developed by the Swedish Institute of Computer Science (SICS). SICStus Prolog has capabilities for Constraint Logic Programming in Finite Domains (CLPFD) that allow several important developments. Most notably, CLPFD allows arithmetic constraints on variables to be introduced into a program, and it can perform arithmetic on these variables even when they are uninstantiated (Pountain 1995). For example, one interval $[S_A,E_A]$ can be constrained to ***overlap*** a second interval $[S_B,E_B]$ using the following constraints (see Section 6.1.1):

$$S_A =< S_B,$$

$$E_A >= S_B,$$

$$E_A =< E_B.$$

A database of constraints is maintained by the *inference layer* of a DIPS agent, and the SICStus Prolog emulator is used as a back-end schedule solver. When a DIPS Resource Agent receives a task request, it posts any new constraints regarding that task to the constraint database and then queries the Prolog constraint management layer for a feasible instance of its local schedule. When the new constraints are propagated through the constraint network, the schedule will reflect these changes.

A DIPS System Agent uses its constraint management layer for a different purpose. Because a DIPS System Agent has no local resources and therefore no local schedule to maintain, it uses the SICStus emulator to generate regional schedules for individual task decompositions.

# 4. HIERARCHICAL MODELING

As mentioned in the introduction, a planning process based on a HTN representation first constructs a plan containing abstract high-level activities and then refines these components in more detail. This process of refinement continues until these high-level activities correspond to the physical actions in the real world. The advantage of this approach is that the feasibility of a plan can be studied incrementally. If a DIPS agent is implementing the above refinement process then domain knowledge of the tasks and their components have to be codified in some language. We provide here some examples of HTN representations (similar to Das, Fox et al. (1997)) used in DIPS.

## 4.1 Compound Goals

A compound task specification used by DIPS has two components: 1) a set of *subtasks* that compose a possible plan for achieving the goal; 2) a set of temporal *constraints* including constraints on the ordering of the subtasks. These subtasks may also be compound themselves. A simplified example task decomposition for a DIPS Satellite Agent is provided below:

```
infrared-picture[Start,End,Filter,Long,Lat] =
        { CameraAgent.picture[Start,End,Filter,Long,Lat];
          RecordingAgent.record[Start,End];
          TransmitterAgent.transmit[Start,End] }
        { task1 span 8; task2 span 2; task3 span 5;
          task1 before task2; task2 before task3;
          task1 during [Start,End]; task3 during [Start,End]; }
```

This defines the high-level goal "infrared-picture" in terms of its subcomponents. Each of these subcomponents may be compound in turn; a DIPS agent can only reason about the immediate sublevel in the HTN. The example decomposition has three subtasks: 1) A Camera Agent takes the picture; 2) A Recording Agent records the picture; 3) A Transmitter Agent transmits the picture to Earth.

The second component of the example contains constraints that relate the subtasks to each other and to the request domain. The minimum duration of each subtask is specified as an integer by the "span" constraints, while the rest of the constraints may be any of those recognized by the DIPS *constraint management layer*. This decomposition plan, a list of subgoals and temporal constraints, is sent through a query to the SICStus Prolog scheduling predicate, which returns a feasible instance of the schedule. (How this instantiation is chosen is described in Section 6.) The agent can then distribute the subtasks to the appropriate subagents with the allocated portions of the original request domain.

## 4.2 Primitive Tasks

Each *primitive task* (or atomic action) in the DIPS scenario world corresponds to a call made to a scripting language such as SCL. These scripts themselves may contain several actions, but due to the fixed nature of a primitive task, these are considered immutable *subatomic* actions. An example of a primitive task specification for a Power Agent is provided below:

```
power-on[Start,End] =
        { "power-on from <Start> to <End>" }
        { task1 span 1 }
```

Here the "plan" for achieving this goal is simply a script call, and the constraints component describes the minimum duration for the task—in this case 1 time unit.

# 5. AGENT COMMUNICATION

Coherence, cooperation and conflict resolution can be improved by carefully designing the amount and type of communication among agents in the form of messages (Patil, Fikes et al. 1992). The information communicated should be relevant, timely and complete (Durfee 1985). Any inter-agent communication in DIPS uses a KQML-compliant format to enhance robustness and modularity.

A DIPS KQML message is considered valid if it has at least the following fields: *sender*, *receiver*, *id* and *path*. Every KQML message also has a *performative* that describes the type of

communication. Most messages also have a *content* field containing an expression describing the purpose of the message.

Agents from the DIPS system currently use only a subset of the standard KQML performatives. The most commonly used are the following: *insert*, *evaluate*, *confirm*, *authorize*, and *sorry*.

Following is a simple example of a goal request sent from a CameraAgent to a FilterAgent:

```
evaluate :content goal(set-filter[125,135])
:id Home-22-Satellite1-0-3-CameraAgent2-0-2
:priority 10 :sender CameraAgent2 :receiver
FilterAgent1
```

The *id* field describes the absolute path followed by this goal request thread from its originating agent to the current node in the HTN, including the index of the decomposition used and the index of the subtask that spawned the current request.

# 6. DECENTRALIZATION AND COORDINATION

The key concept of the DIPS system is to incrementally partition the scheduling problem into smaller independent subproblems of increasing granularity which can then be solved in parallel. While this may sacrifice global completeness and optimality in the search for a feasible plan, the distributed scheduling approach greatly reduces the complexity of a large-scale multiple-resource scheduling problem.

## 6.1 Least Commitment Scheduling

The important development made by the use of the *constraint management layer* is that relational constraints on time and resource consumption can be specified in the scheduling process. This allows the full and precise description of plans and task decompositions without any specification of actual start times. Empowering a DIPS agent with this mentality of *least commitment* allows reactive planning and dynamic rescheduling; the Prolog scheduler will allow local changes to be made while adhering to the original global constraints.

### *6.1.1 Recognized Temporal Constraints*

As proposed by Allen (1984), we will recognize 7 basic temporal relations between two actions. These relations for two actions, *A* and *B* with intervals $[S_A,E_A]$ and $[S_B,E_B]$ respectively, are shown below in Table 1. Any other relation can be expressed as the inverse of one of these.

| | |
|---|---|
| 1. A **before** B | $E_A<S_B$, |
| 2. A **during** B | $S_B<S_A$, $E_A<E_B$ |
| 3. A **overlaps** B | $S_A<S_B$, $S_B<E_A<E_B$ |
| 4. A **equals** B | $S_A=S_B$, $E_A=E_B$ |
| 5. A **meets** B | $E_A=S_B$ |
| 6. A **starts** B | $S_A= S_B$, $E_A<E_B$ |
| 7. A **finishes** B | $S_B< S_A$, $E_A= E_B$ |

**Table 1: Recognized Temporal Constraints**

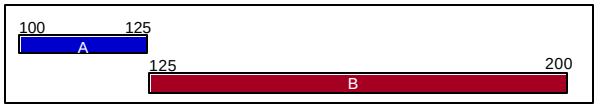
The first three constraints, **before, during,** and **overlaps**, are the three basic types of relations. Constraining two actions with these relations allows some flexibility in the instantiation of real start and end times. The last four can be considered special cases of the basic types, which all involve reasoning about specific start or end times for the actions and do not allow any flexibility for one event once the other has been bound to a real interval.

## 6.2 Domain Distribution Algorithm

In contrast to previous distributed planning and scheduling systems such as DAS (Burke 1991), DIPS employs a *domain distribution algorithm* that typically allows a feasible plan to be found after only the first coordination effort, therefore greatly decreasing the need for inter-agent negotiation during the scheduling process. As an extension to the *least commitment* approach, each node in the DIPS HTN leaves as much flexibility in the next sublevel as possible when coordinating the domain distribution. The basic underlying algorithm: when choosing a feasible schedule (partial ordering) of subtasks for a given compound goal, allocate to each subtask as *generous* (large) a time interval as possible without allowing global constraints to be violated when the subtasks are instantiated on the next level in the HTN.

The simplest solution is to allocate to each subtask a portion of the whole domain the size of which is relative to the duration of the subtask. This solution works perfectly well when dealing only with the *ordering* of tasks (when the only type of constraint being used is the ***before*** constraint). Consider the following example decomposition: Two tasks, *A* with interval $[S_A,E_A]$ and *B* with interval $[S_B,E_B]$, are to be scheduled during the request domain [100,200]. The only constraints on the tasks are their durations *A* ***spans*** *10*, *B* ***spans*** *30*, and *A* ***before*** *B*. The following schedule could be generated that would satisfy the constraints without using a *generous* distribution:


Using the DIPS *generous* approach to domain distribution, however, these subtask domains would be as large as possible in order to provide flexibility at lower levels in the HTN (note that the ratio of the domain sizes *A:B* remains the same)



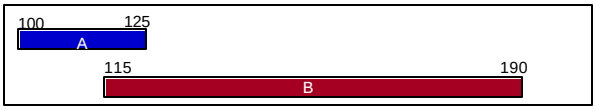

The second instance of the schedule will allow flexibility in the instantiation of both subtasks when they are distributed to the subagents. In both cases the original constraints are fulfilled, but the first example is more likely to require inter-agent negotiation

(and thus to some extent a global search) when local resources are overscheduled during certain intervals. The second example demonstrates the *least commitment* paradigm by relaxing the constraints on the tasks as much as possible.

### 6.2.1 Distribution Conflicts

Applying a *generous* domain distribution algorithm is clearly simple in cases of basic ordering between tasks (only using the ***before*** constraint). In order to distribute portions of a scheduling problem that includes any of the other temporal relations, a system for managing global constraints must be developed.

Let us consider a slightly more complicated example regarding the same tasks *A* and *B*, except that they are related with the following constraint: *A* ***overlaps*** *B*. (See Table 1.) A flawed *generous* approach might produce this schedule instead:




Consider the possible instantiations for each subtask on the next level of the HTN. It is entirely possible that *A* could start and finish (e.g., [100,110]) before the start of *B* (e.g., [115,145])

The DIPS algorithm solves this problem by treating the constraints ***during*** and ***overlaps*** as special cases. The Prolog decomposition predicate increases the size of the domain of each subtask within the following boundaries: if one subtask is related to another by one of the inflexible constraints, (***meets, starts, finishes, equals***), the domain size is left equal to the duration of the subtask; if the subtask is constrained to other subtasks only by the ***before*** relation, the domain size is expanded proportionally to the duration of the subtask; if there exist any constraints relating this subtask to another with ***during*** or ***overlaps***, several new variables are introduced in order to reason about the constrained intervals. An interval $[S_B,E_B]$ allocated for task *B* has three components at the decomposition level: the domain start $S_B$, the domain end $E_B$, and the expected duration spanned by the task $D_B$. By definition, $D_B =< E_B - S_B$.

For every task *A* that is constrained to ***overlap*** *B*, the size of the domains of *A* and *B* may be increased while *A* cannot be started anywhere within its domain such that it does not overlap *B*, and vice-versa. Compare this formula with the simple version of *overlaps* in Section 3.3.

$$S_A+D_A>=E_B-D_B,$$

$$S_B>=E_A-D_A,$$

$$E_A<=S_B+D_B.$$

For every task *A* that is constrained to be ***during*** *B*, the size of the domains of *A* and *B* may be increased while *B* cannot be started anywhere within its domain such that it does not fully include the domain of *A*.

$$E_B-S_A=<D_B,$$

$$E_A-S_B=<D_B.$$

Again examining the above intervals *A*=10 and *B*=30, the improved *generous* approach might produce the following schedule instead. This schedule still provides flexibility on the next level but without the possibility of global constraint violation:


## 7. IMPLEMENTATION

The current system for simulating a scenario uses one main Launcher application written in Java to start each DIPS agent as an independent thread with its own initialization file of capability insertions in the form of KQML messages. An interface to the Launcher allows customization of the scenario at runtime.

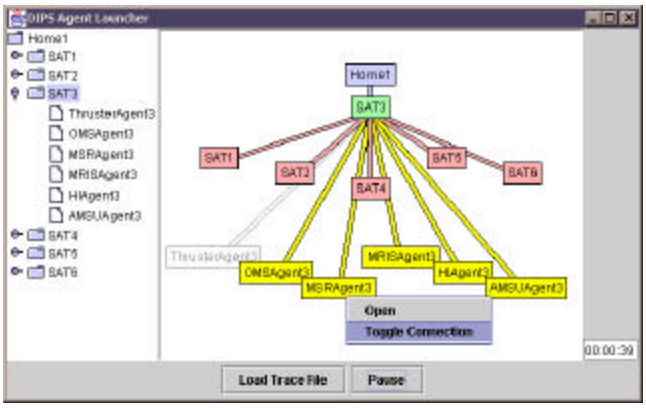



**Figure 3: DIPS Launcher Interface**

In this example view, *SAT3*, has 5 Satellite siblings (middle row) and 6 child agents (bottom row), each of which is a DIPS Resource Agent. As shown above the link between certain agents can be disabled to simulate a communications lapse. Through this interface, trace files of inter-agent goal requests (in KQML format) are introduced into the DIPS community. Whenever two task requests conflict on a local resource, the DIPS System Agent that originated the request will automatically attempt to reschedule that subtask on the next cheapest resource.

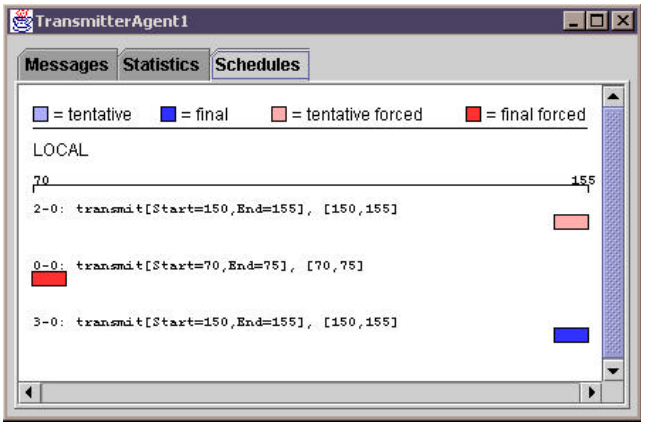



**Figure 4: Agent Local Schedule**

Figure 4 shows the local schedule of a DIPS Transmitter agent, containing three locally scheduled tasks, two of which (*2-0*, *3-0*) are conflicting, but only one of which, *2-0*, is high priority (or *forced*). Because task *2-0* is still tentative, the agent has allowed a lower priority task to be given the same time interval. When the task is authorized (finalized by the originating agent), it will force out any conflicting requests due to its high priority. Thus in Figure 5 below the conflicting request *3-0* has been removed from the schedule once *2-0* was authorized.

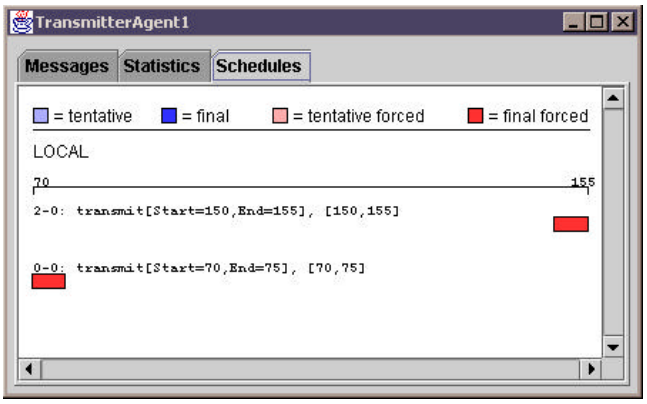



**Figure 5: Authorized Local Schedule**

We can also see the resulting messages sent to *Home1* in Figure 6. Notice that the high priority task, *2-0*, was successfully authorized, while the conflicting low-priority task, *3-0*, could no longer be scheduled consistently on all necessary resources, and a *sorry* message was therefore returned up the HTN to the *Home1* DIPS agent.

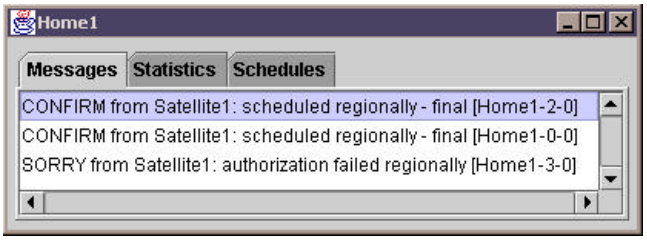



**Figure 6: Task Authorization Replies**

In addition to dealing with request conflicts, we submit a set of similar task requests within a certain time interval in order to examine the performance of the DIPS-based community under heavy request and scheduling loads. Communications travel in both directions through the hierarchy, including *confirm* and *sorry* replies from subagents pertaining to high-level goal requests. An example scenario consisting of 4 System agents and 14 Resource agents (a total of 18 HTN nodes) can distribute and schedule 35 high-level goal requests corresponding to 280 *primitive tasks* in under 5 minutes, even when running on a single CPU.

## 8. CONCLUSION AND FUTURE WORK

In this paper, we have demonstrated how our distributed approach to planning and scheduling helps to achieve high-level goals and thereby enhances spacecraft autonomy. A hierarchical syntax has been adopted for representing domain knowledge of task decompositions and employed to solve the task decentralization problem. A constraint propagation paradigm has been employed for the required planning and scheduling tasks performed by an autonomous agent, and an innovative system for the decentralization of a global scheduling problem has been developed and employed with promising results. Priority-based scheduling has been implemented, and a hierarchical inter-agent confirmation/authorization system was used for global goal coordination.

The relative speed with which new requests can be processed indicates two areas of success by the DIPS system: 1) efficient multiple-resource scheduling in a distributed environment by reduction of the complexity of the global scheduling problem; and 2) implementation of a *generous* domain distribution algorithm that minimizes inter-agent negotiation. The global scheduling problem in our multi-resource scheduling example would be computationally difficult for a single scheduling agent to solve as a whole; yet after decentralization through the DIPS HTN, the total search space was reduced exponentially, and a solution was produced with relative efficiency. A solution is found with much greater efficiency while still satisfying global constraints.

The efforts of the DIPS system currently focus on schedule creation and maintenance rather than on schedule optimization due to the dynamic real-time nature of the application. Further efforts include the use of CPU idle time for replanning and schedule optimization based on cost utilities. Ongoing efforts also include integration with SCL to enhance the real-time and real-world applicability of the DIPS system. The initial scenario will be augmented by incorporating actual capabilities of real satellites such as NASA's planned nanosatellite constellations in order to model real problems that may arise. Actual task decompositions of standard observing sequences will form the core of the task requests.

**Acknowledgements:** The authors would like to thank Paul Gonsalves and Dan Grecu of Charles River Analytics for their contributions to the planning and scheduling algorithm and Allan Posner of ICS for his contributions to the scenario. This work was performed under contract NAS5-99168 with NASA Goddard Space Flight Center.

## 9. REFERENCES

[1] Allen, J. F. (1984). "Towards a General Theory of Action and Time." Artificial Intelligence 23.

[2] Boddy, M. (1994). "Temporal reasoning for planning and scheduling." SIGART Bulletin 4(3).

[3] Burke, P., and Prosser, P. (1991). "A distributed asynchronous system for predictive and reactive scheduling." Artificial Intelligence in Engineering 6: 106-124.

[4] CACM (1994). Intelligent Agents - Communication of the ACM, ACM Press.

[5] Chaib-draa, B. M., Mandiau, R., and Millot, P. (1992). "Trends in distributed artificial intelligence." Artificial Intelligence Review 6: 35-66.

[6] Chapman, D. (1987). "Planning for Conjunctive Goals." Artificial Intelligence 32: 333-378.

[7] Conry, S. M., R. and Lesser, V.R. (1988). Multiagent negotiation in distributed planning. Reading in Distributed Artificial Intelligence. A. a. G. Bond, L., Morgan Kaufmann: 367-384.

[8] Currie, K. a. T., A. (1991). "The Open Planning Architecture." Artificial Intelligence 52(1): 49-86.

[9] Das, S. K., J. Fox, et al. (1997). Decision Making and Plan Management by Autonomous Agents: Theory, Implementation, and Applications. First International Conference on Autonomous Agents, California.

[10] Dean, T., Firby, R. J., and Miller, D. (1988). "Hierarchical planning involving deadlines, travel time, and resources." Computational Intelligence 4: 381-398.

[11] Durfee, E. H., Lesser, V.R., and Corkill, D.D. (1985). Increasing coherence in a distributed problem-solving network. 8th International Joint Conference on Artificial Intelligence.

[12] Ferguson, I. A. (1992). Touring Machines: AN Architecture for Dynamic, Rational. Mobile Agents, Computer Laboratory, University of Cambridge.

[13] Fikes, R. a. N., N. (1971). "STRIPS: A new approach to the application of theorem proving to problem solving." Artificial Intelligence 2: 189-208.

[14] Genesereth, M. R. and S. P. Ketchpel (1994). "Software Agents." Communications of the ACM 37(7): 48-53.

[15] Georgeff, M. (1983). Communication and interaction in multi-agent planning. AAAI.

[16] Georgeff, M. P. (1987). "Planning." Annual Review in Computer Science 2: 359-400.

[17] Interface & Control Systems, Inc. (1999). http://www.sclrules.com/.

[18] LePape, C. (1990). Constraint propagation in planning and scheduling.

[19] McAllester, D. a. R., D. (1994). Systematic nonlinear planning. AAAI-94.

[20] Muller, J. P. a. P., M. (1994). Modeling interacting agents in dynamic environments. 11th European Conference on Artificial Intelligence.

[21] Muscettola, N. (1994). Integrating planning and scheduling. Intelligent Scheduling. M. a. Z. Fox, M., Morgan Kaufmann.

[22] Patil, R. S., R. E. Fikes, et al. (1992). The DARPA Knowledge Sharing Effort: Progress Report. Proceedings of Knowledge Representation and Reasoning (KR%R-92). C. Rich, W. Swartout and B. Nebel: 777-788.

[23] Pountain, D. (1995). "Constraint Logic Programming." Byte.

[24] Rosenschein, J. S. (1982). Synchronization of multi-agent plans. National Conference on Artificial Intelligence.

[25] Sacerdoti, E. (1974). "Planning in a hierarchy of abstraction spaces." Artificial Intelligence 5: 115-135.

[26] Simmons, R. (1991). "Coordinating planning, perception, and action for mobile robots." SIGART Bulletin 2: 156-159.

[27] Tate, A. (1977). Generating project networks. 5th International Joint Conference on Artificial Intelligence.

[28] Wilkins, D. E. (1988). Practical Planning: Extending the Classical AI Planning Paradigm, Morgan Kaufmann Publishers.